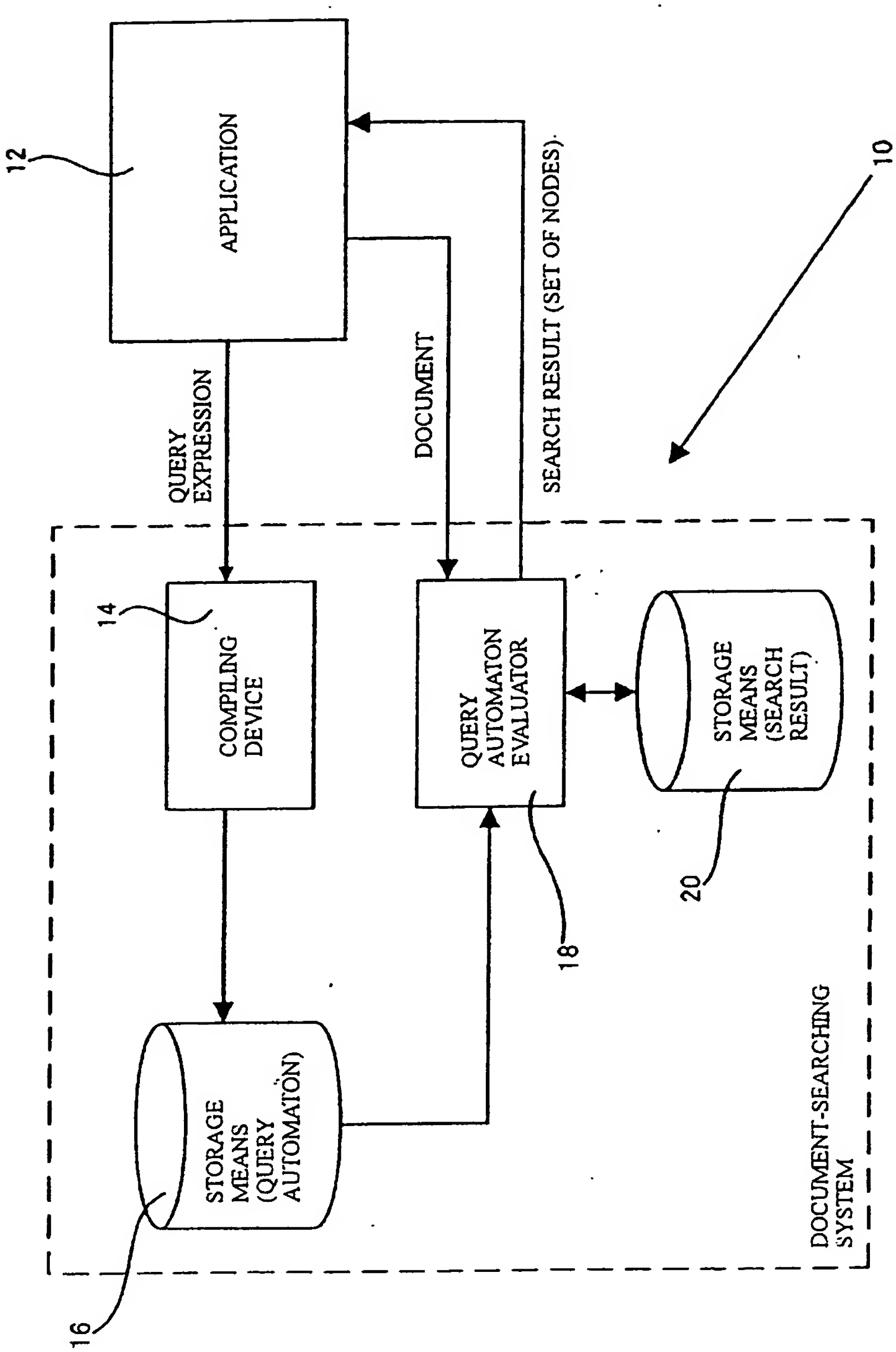


APPLICATION
12
QUERY EXPRESSION
DOCUMENT
SEARCH RESULT (SET OF NODES)
10
COMPILING DEVICE
14
QUERY AUTOMATON EVALUATOR
18
STORAGE MEANS (SEARCH RESULT)
20
STORAGE MEANS (QUERY AUTOMATON)
16
DOCUMENT-SEARCHING SYSTEM

Figure 1

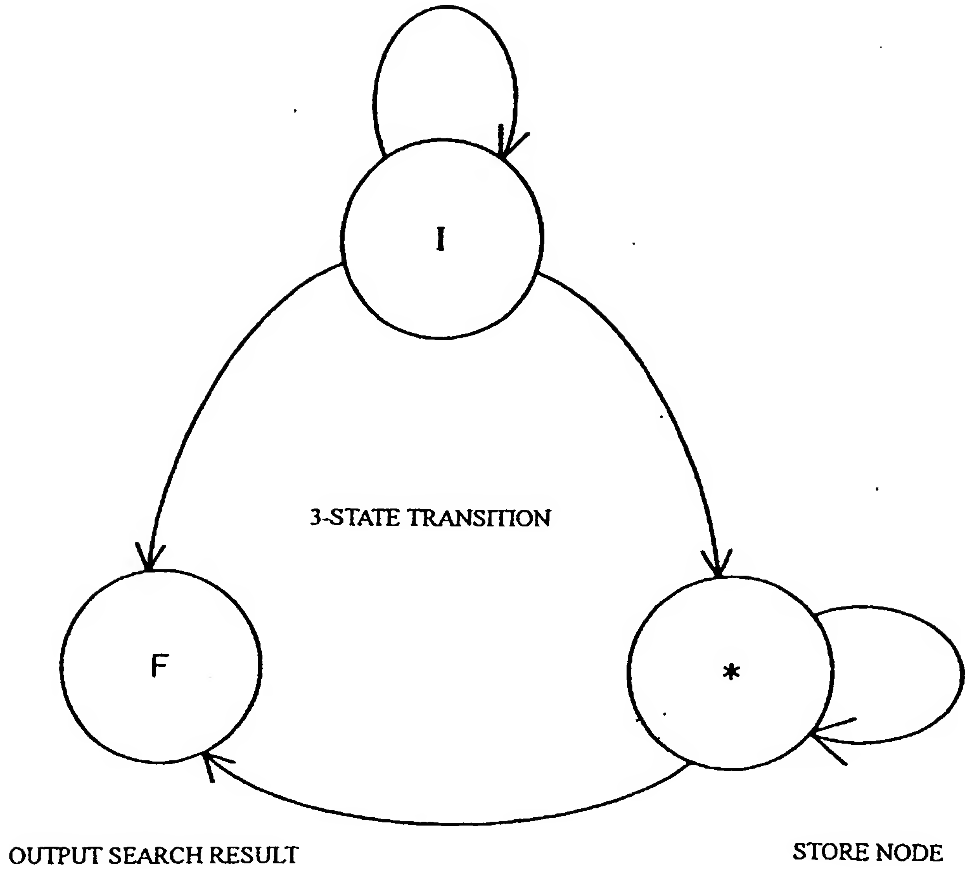


Figure 2

```
<html>
  <head><title>Sample</title></head>
  <body>
    <div>
      <p>A sample document</p>
    </div>
    <p>A sample paragraph in div</p>
  </body>
</html>
```

Figure 3

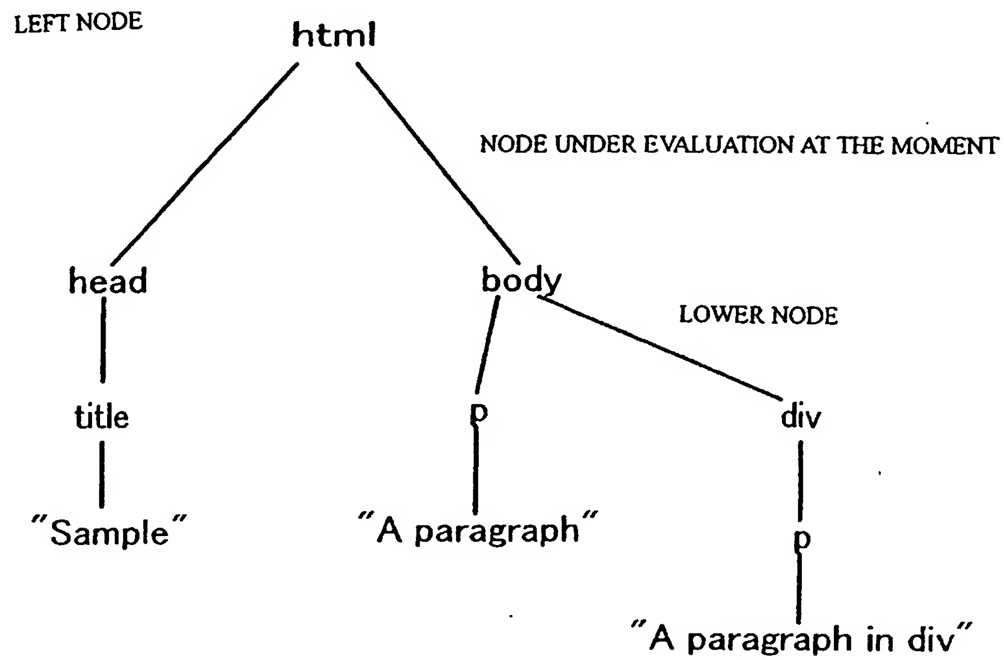


Figure 4

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	I, F	$\neg$p	0, 1	0, 1
1	*, I, F	any	0, 1	0, 1

Figure 5

QUERY AUTOMATON FOR //p[not(ancestor::div)]

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	I, F	¬p	0, 1	0, 1
0	I, F	div	1	2
1	*I, F	any	2	0,1
2	I	any	0,1	2

(a)

QUERY AUTOMATON FOR //*[.//div and.//p]

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	F	¬div	0, 1	2
0	F	¬p	0, 1	3
1	*, I, F	any	0, 1	0, 1
2	I	¬div	2	2
3	I	¬p	3	3

(b)

Figure 6

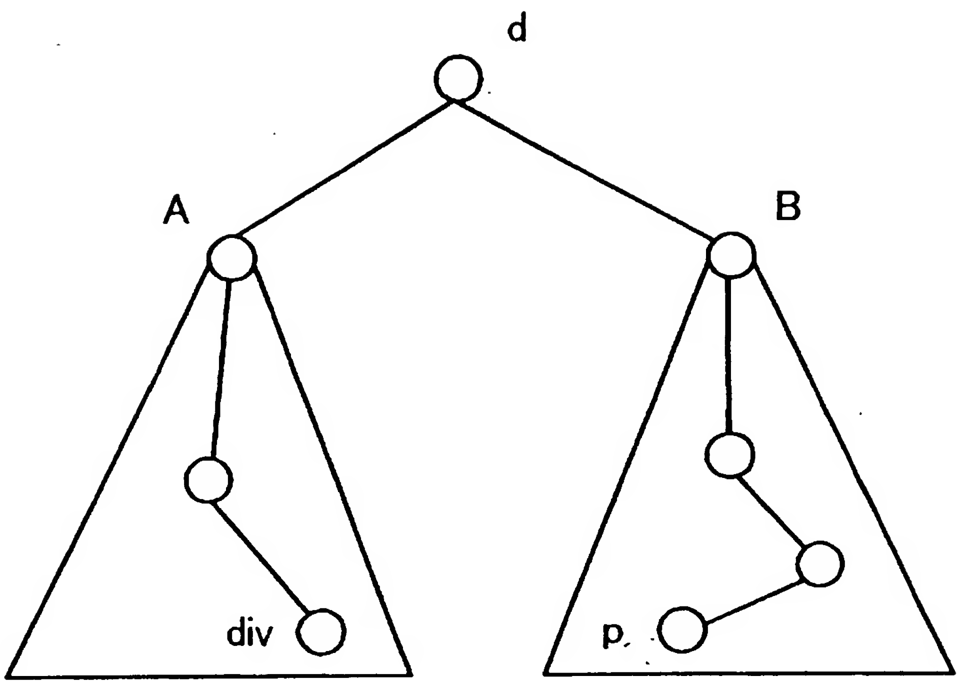


Figure 7

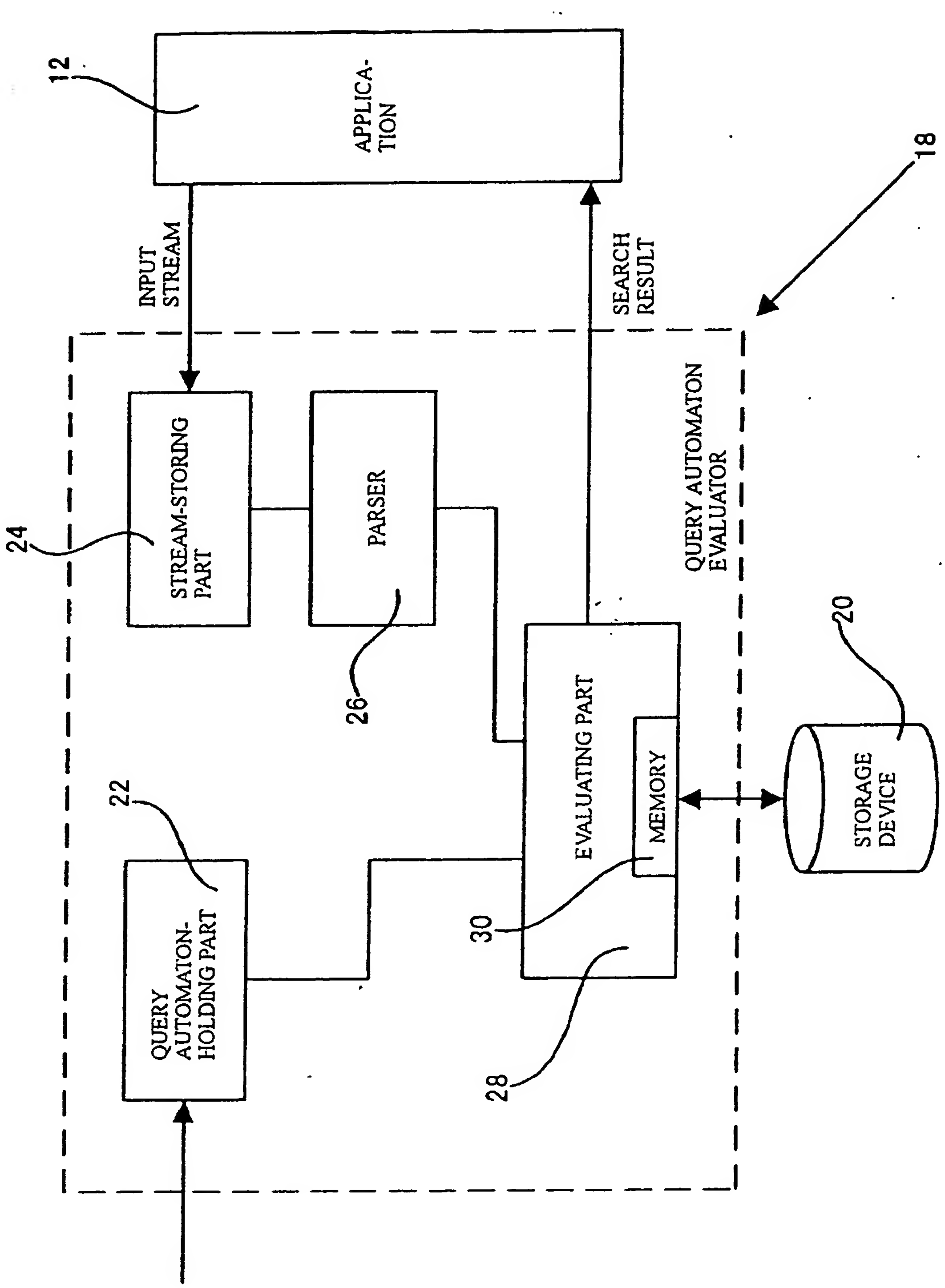
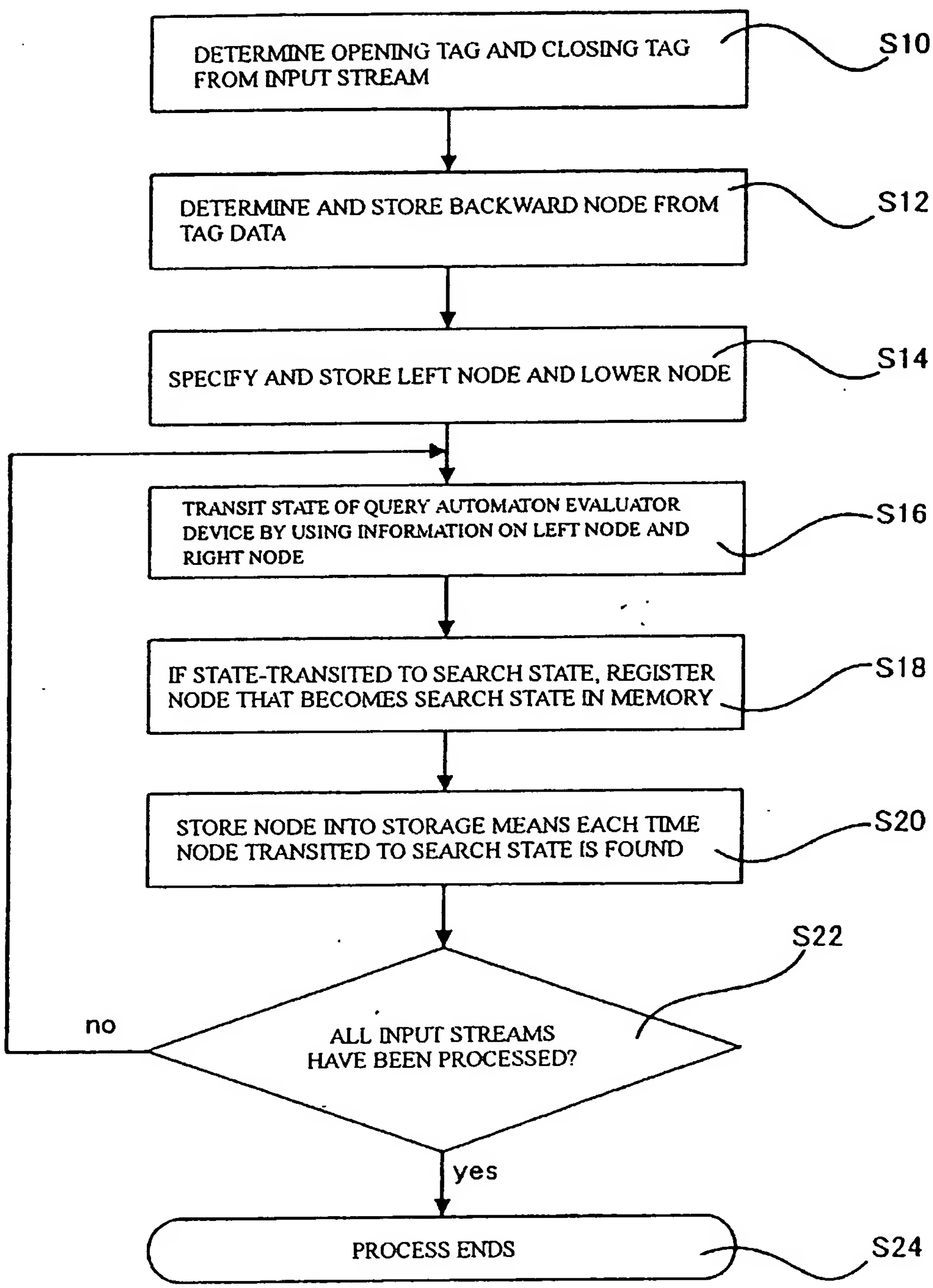
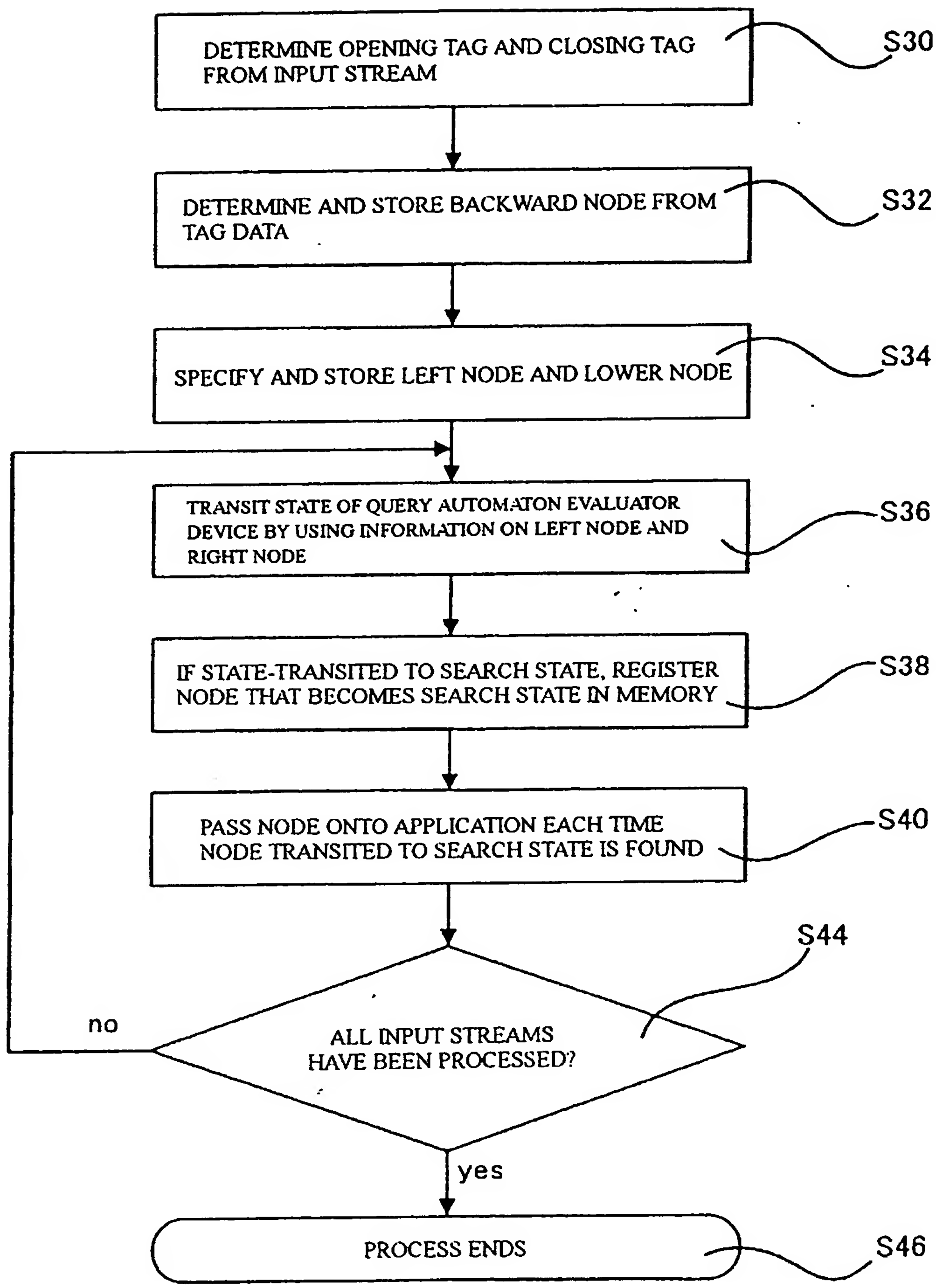

12
APPLICA-
TION
INPUT
STREAM
SEARCH
RESULT
18
24
STREAM-STORING
PART
PARSER
26
QUERY AUTOMATON
EVALUATOR
22
QUERY
AUTOMATON-
HOLDING PART
EVALUATING PART
MEMORY
30
28
20
STORAGE
DEVICE

Figure 8



DETERMINE OPENING TAG AND CLOSING TAG FROM INPUT STREAM
S10
DETERMINE AND STORE BACKWARD NODE FROM TAG DATA
S12
SPECIFY AND STORE LEFT NODE AND LOWER NODE
S14
TRANSIT STATE OF QUERY AUTOMATON EVALUATOR DEVICE BY USING INFORMATION ON LEFT NODE AND RIGHT NODE
S16
IF STATE-TRANSITED TO SEARCH STATE, REGISTER NODE THAT BECOMES SEARCH STATE IN MEMORY
S18
STORE NODE INTO STORAGE MEANS EACH TIME NODE TRANSITED TO SEARCH STATE IS FOUND
S20
S22
no
ALL INPUT STREAMS HAVE BEEN PROCESSED?
yes
PROCESS ENDS
S24

Figure 9



DETERMINE OPENING TAG AND CLOSING TAG FROM INPUT STREAM
S30
DETERMINE AND STORE BACKWARD NODE FROM TAG DATA
S32
SPECIFY AND STORE LEFT NODE AND LOWER NODE
S34
TRANSIT STATE OF QUERY AUTOMATON EVALUATOR DEVICE BY USING INFORMATION ON LEFT NODE AND RIGHT NODE
S36
IF STATE-TRANSITED TO SEARCH STATE, REGISTER NODE THAT BECOMES SEARCH STATE IN MEMORY
S38
PASS NODE ONTO APPLICATION EACH TIME NODE TRANSITED TO SEARCH STATE IS FOUND
S40
S44
no
ALL INPUT STREAMS HAVE BEEN PROCESSED?
yes
PROCESS ENDS
S46

Figure 10

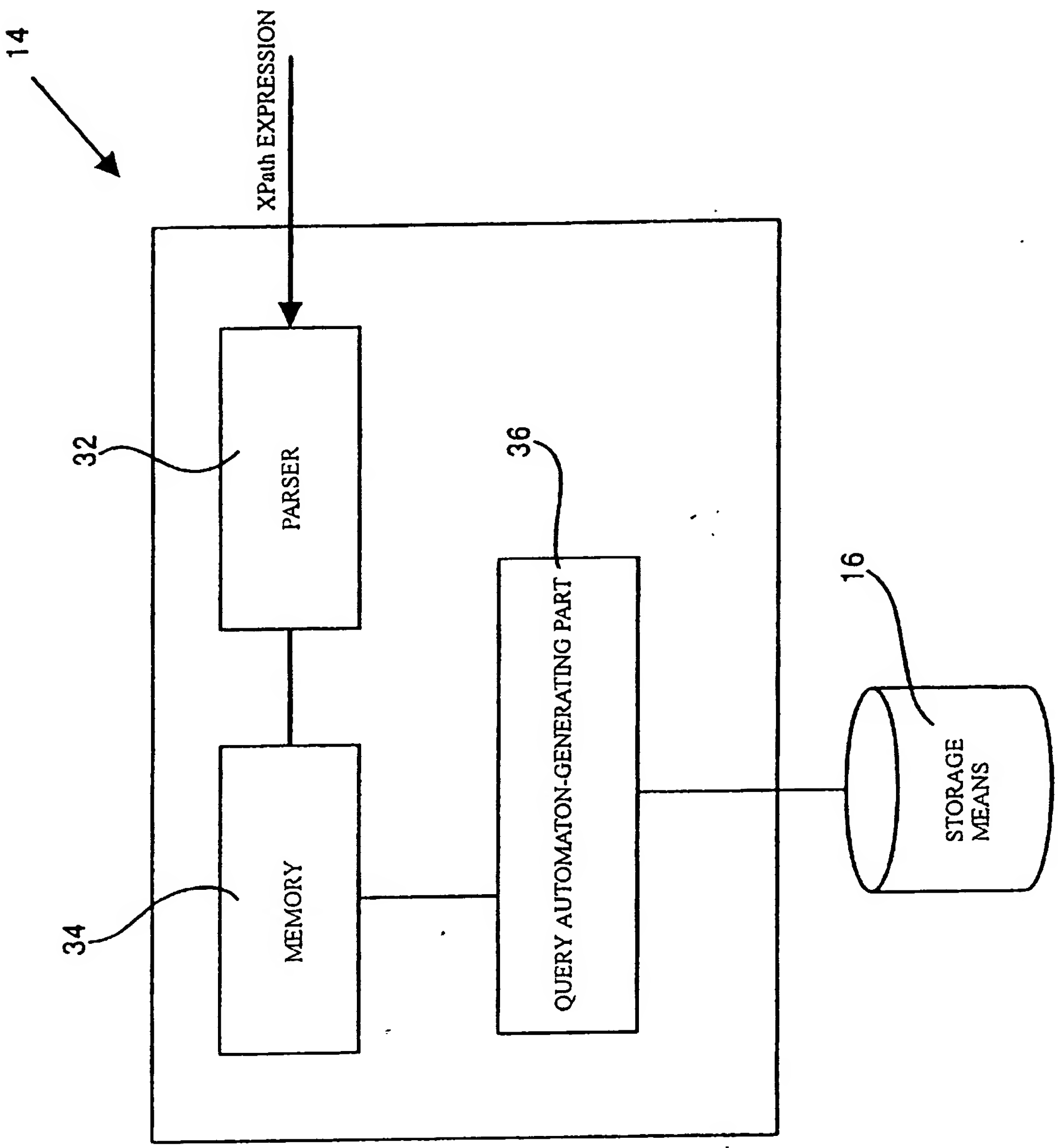

14
XPath EXPRESSION
32
PARSER
34
MEMORY
36
QUERY AUTOMATON-GENERATING PART
16
STORAGE MEANS

Figure 11

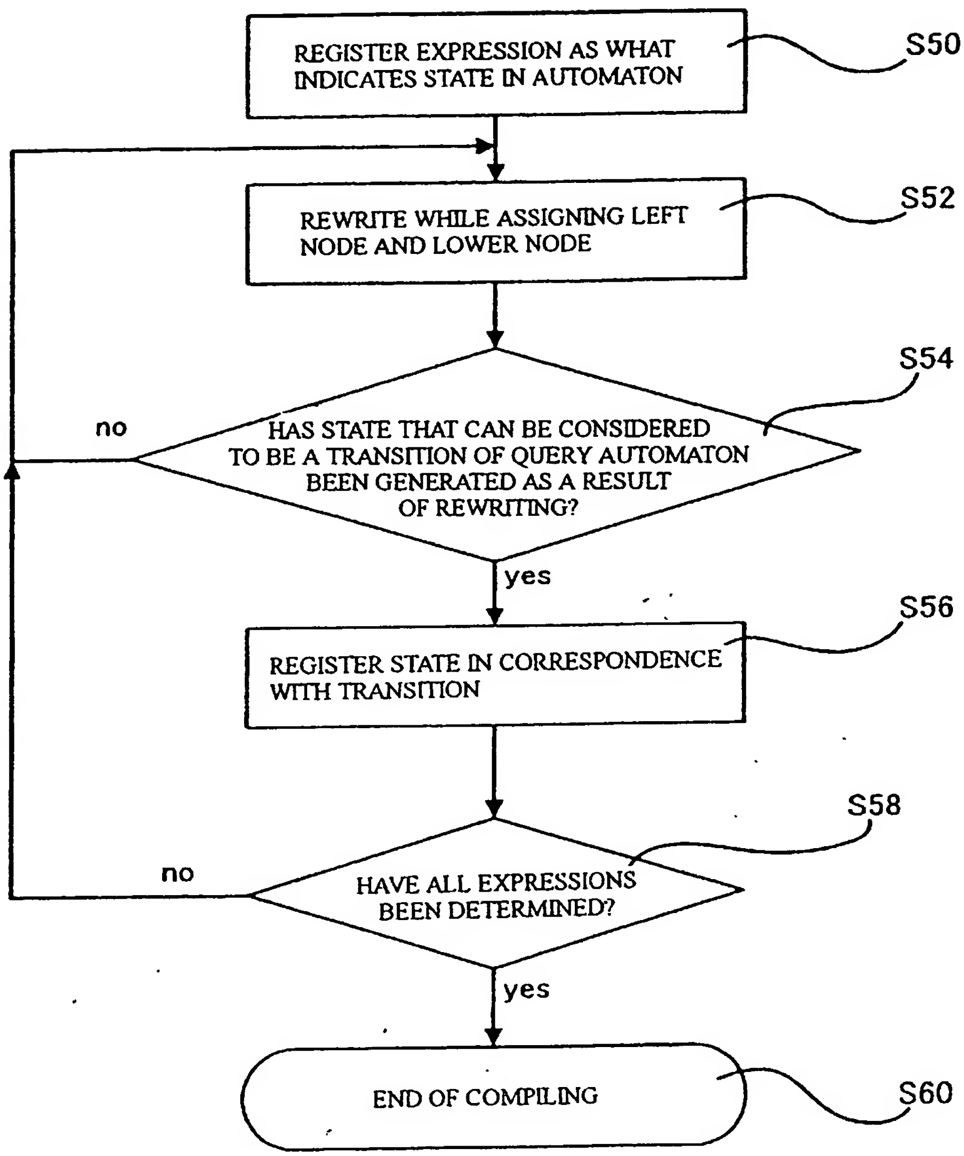

REGISTER EXPRESSION AS WHAT INDICATES STATE IN AUTOMATON
S50
REWRITE WHILE ASSIGNING LEFT NODE AND LOWER NODE
S52
HAS STATE THAT CAN BE CONSIDERED TO BE A TRANSITION OF QUERY AUTOMATON BEEN GENERATED AS A RESULT OF REWRITING?
S54
no
yes
REGISTER STATE IN CORRESPONDENCE WITH TRANSITION
S56
HAVE ALL EXPRESSIONS BEEN DETERMINED?
S58
no
yes
END OF COMPILING
S60

Figure 12

```
proc eval(v)
  case v
    u <σ> w </σ> → Ξ := { <q, {x | ∀q', q''. q ∈ δ(σ, q', q'')
                                  ⇒ ∃V. x ∈V ∧ (<q', V> ∈eval(u) ∨
<q'', V> ∈eval(w)) }> | q ∈ Q }
    s → Ξ := {<q, s> | q ∈ I }
  esac
  for <q, x> ∈ Ξ
    if (q ∈Θ) Ξ := (Ξ \ {<q, V>}) ∪ {<q, V ∪ {v}>} fi
  rof
  return Ξ
corp
```

Figure 13

```html
<html>
  <head><title>Sample</title></head>
  <body>
    <p>A paragraph</p>
    <div>
        <p>A paragraph in div</p>
    </div>
  </body>
</html>
```

Figure 14

$$\mathrm{eval}(\text{"Sample"}) = \left\langle \{0, 2\}, \text{"Sample"} \right\rangle$$

Figure 15

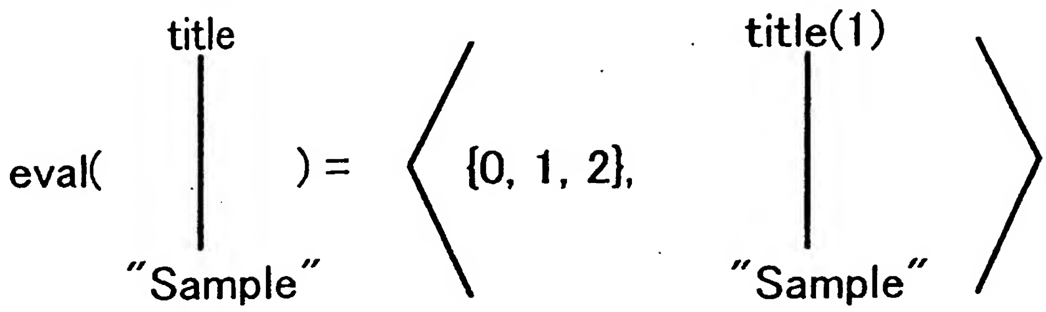


Figure 16

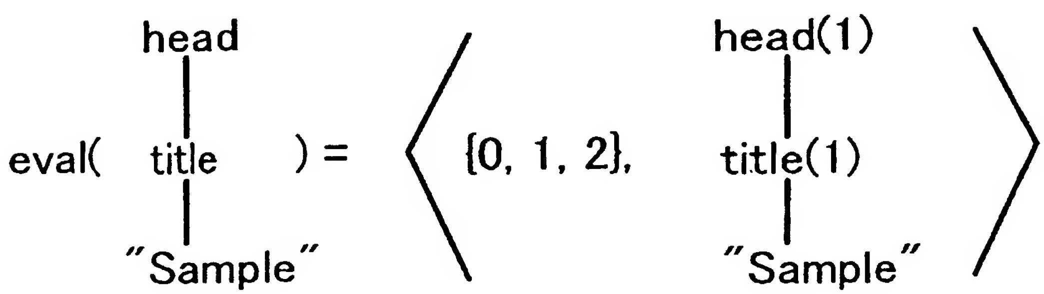


Figure 17

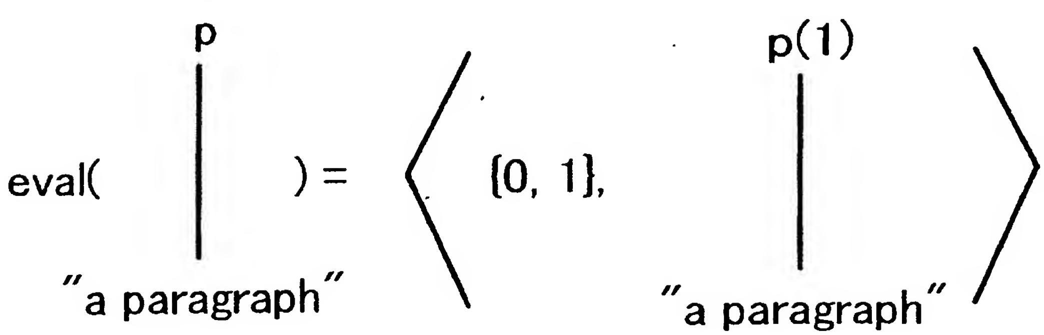
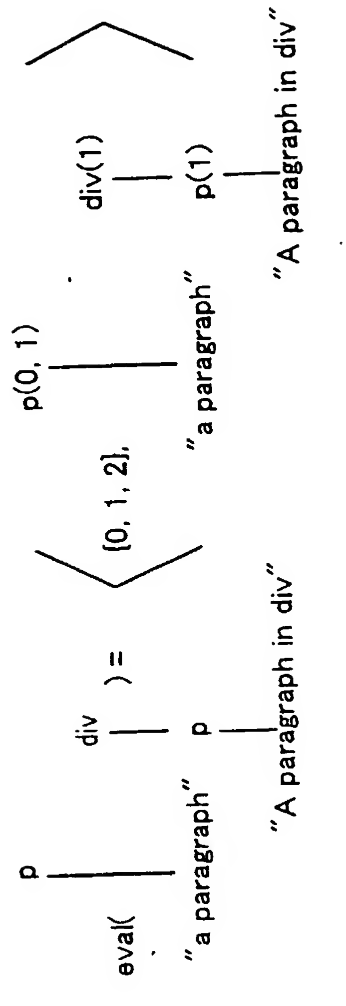


Figure 18



eval(
p
"a paragraph"
div
p
"A paragraph in div"
) =
{0, 1, 2},
p(0, 1)
"a paragraph"
div(1)
p(1)
"A paragraph in div"

Figure 19

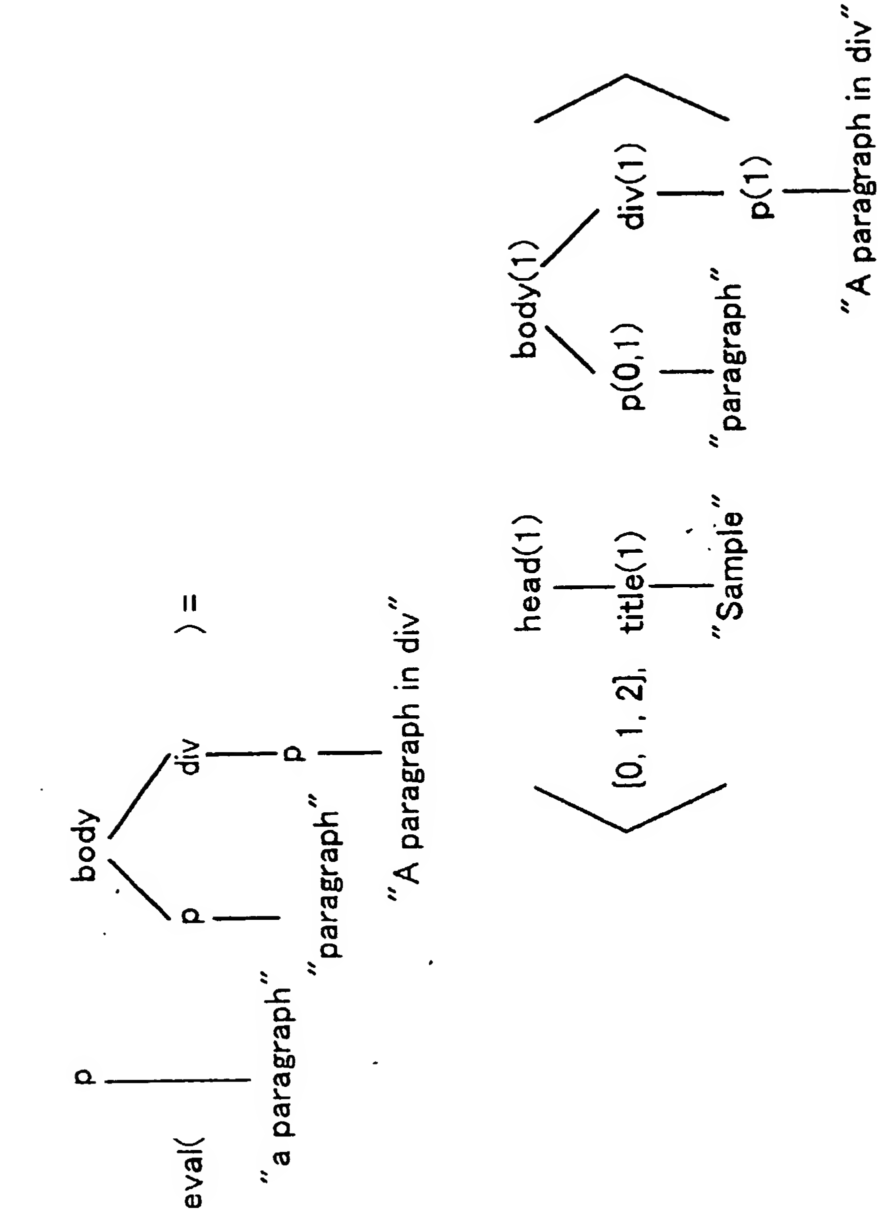
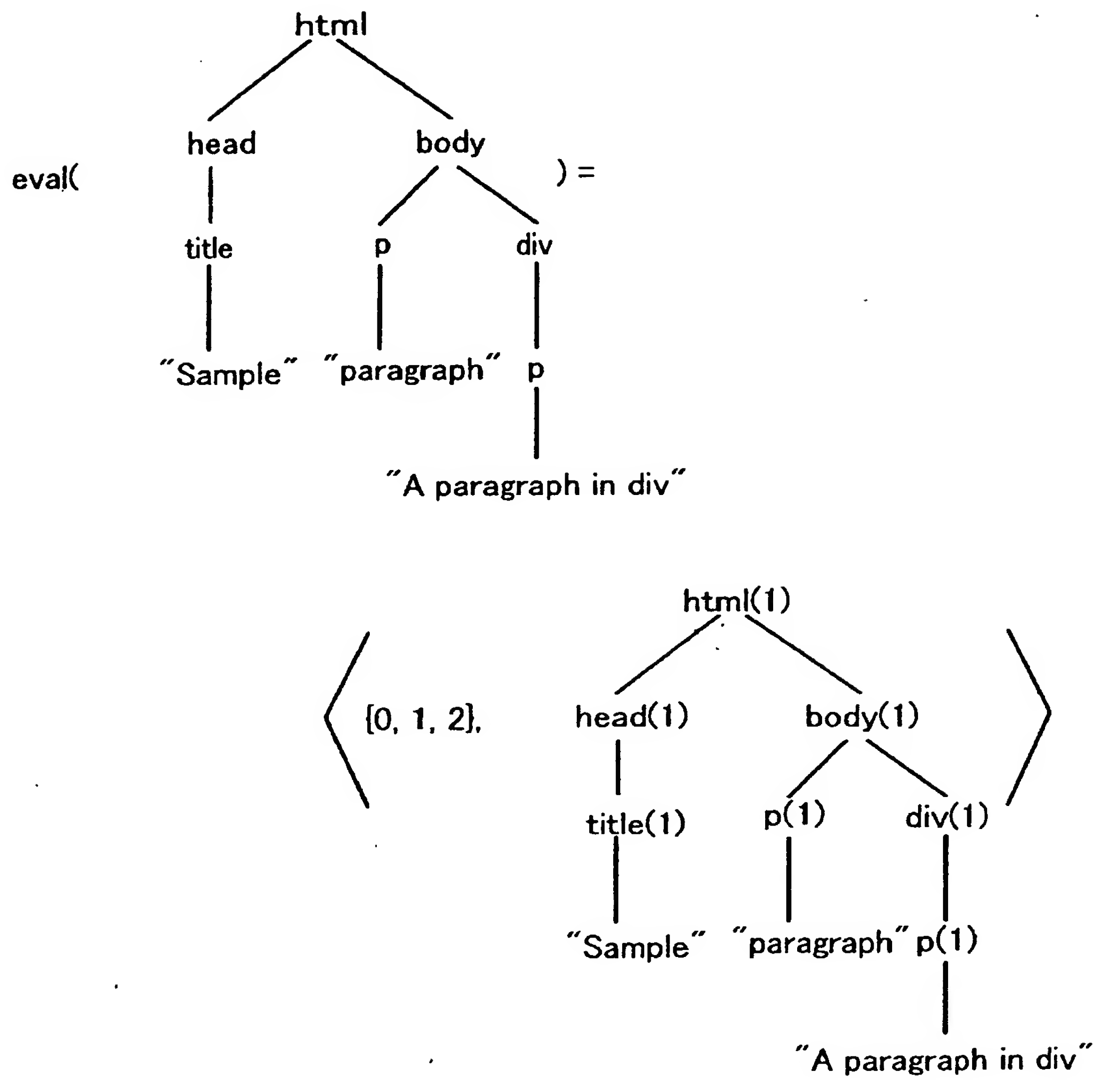

eval(
p
"a paragraph"
body
p
"paragraph"
div
p
"A paragraph in div"
) =
[0, 1, 2],
head(1)
title(1)
"Sample"
body(1)
p(0,1)
"paragraph"
div(1)
p(1)
"A paragraph in div"

Figure 20



eval(
html
head
body
title
p
div
"Sample"
"paragraph"
p
"A paragraph in div"
) =
{0, 1, 2},
html(1)
head(1)
body(1)
title(1)
p(1)
div(1)
"Sample"
"paragraph"
p(1)
"A paragraph in div"

Figure 21

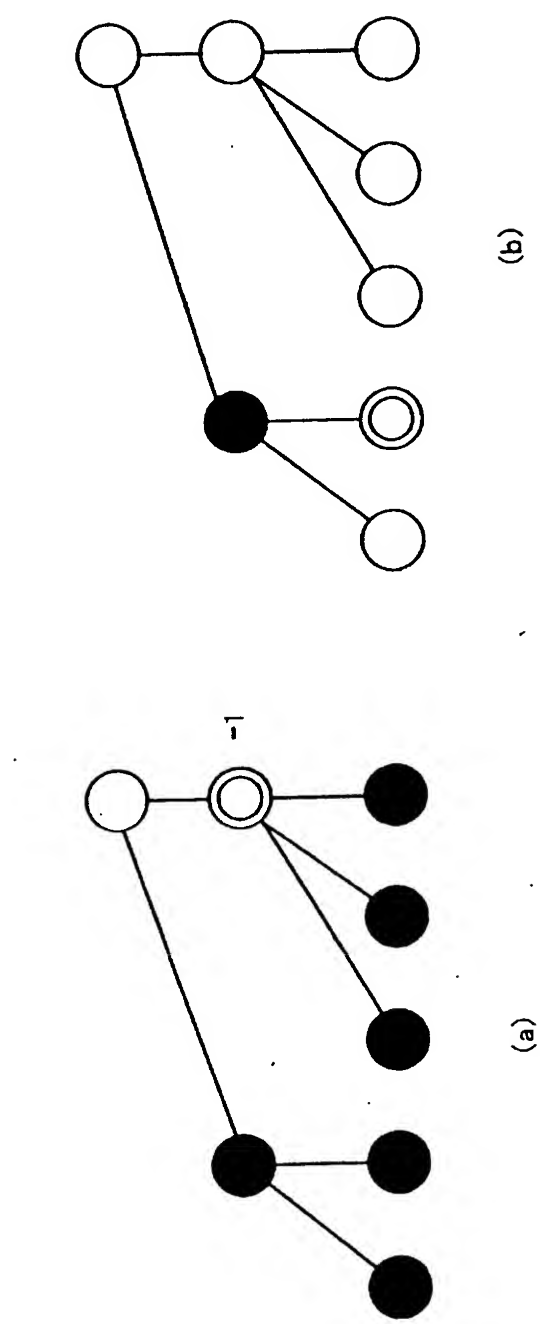


Figure 22

REACHED STATE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
$\neg a \wedge 1[1;2](\neg a \vee *) \wedge 2[1;2](\neg a \vee *)$	$\neg p$	$[1;2](\neg a \vee *)$	$[1;2](\neg a \vee *)$
$* \wedge 1[1;2](\neg a \vee *) \wedge 2[1;2](\neg a \vee *)$	any	$[1;2](\neg a \vee *)$	$[1;2](\neg a \vee *)$

Figure 23

REACHED STATE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	$\neg p$	0, 1	0, 1
1	any	0, 1	0, 1

Figure 24

REACHED STATE	TYPE	CONDITION FOR TRANSITION	STATE OF LEFT NODE	STATE OF LOWER NODE
0	[, F	$\neg p$	0, 1	0, 1
1	*, [, F	any	0, 1	0, 1

Figure 25

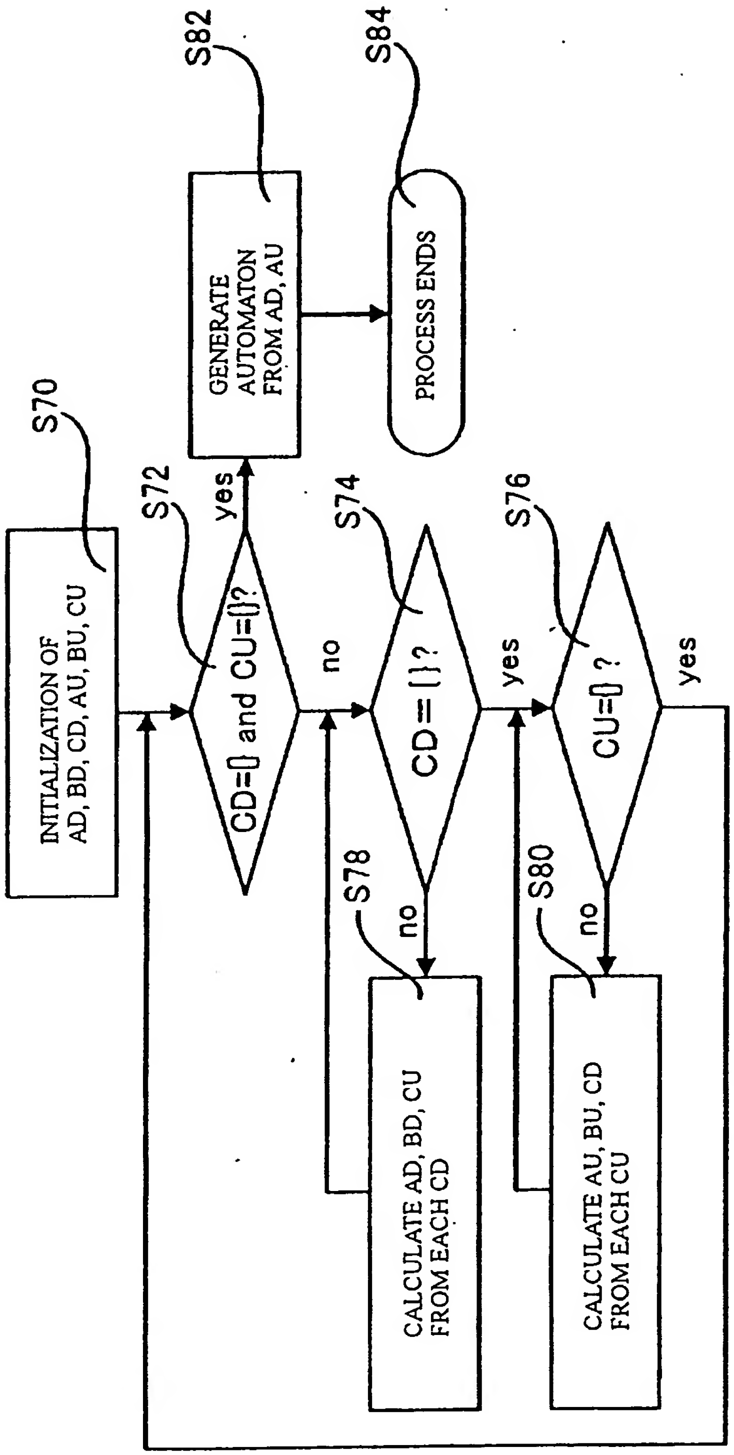

S70
INITIALIZATION OF AD, BD, CD, AU, BU, CU
S72
CD=[] and CU=[]?
yes
no
S74
CD= [ ] ?
yes
no
S76
CU=[] ?
yes
no
S78
CALCULATE AD, BD, CU FROM EACH CD
S80
CALCULATE AU, BU, CD FROM EACH CU
S82
GENERATE AUTOMATON FROM AD, AU
S84
PROCESS ENDS

Figure 26

```
proc main(ϕ)
  Q := P := {T}, S := Resolved := {}
  AD := BD := BU := CU := CD := {}
  AU := {<T, T, T>, <T, T, ε>, <T, ε, T>, <T, ε, ε>}
  T.succ1 := T.succ2 := T.prec1 := T.prec2 := ε.prec1 := ε.prec2 := {}
  S-extend(T, {ϕ})
  while CD ≠ {} or CU ≠ {}
    for <q0, q1, q2> ∈ CD
      CD := CD ╲ <q0, q1, q2>
        X1 := S-extend(q1, q0.succ1)
        X2 := S-extend(q2, q0.succ2)
        for <_, q1', q2'> ∈ {q0} × X1 × X2 ╲ AD ╲ BD
          if ¬∃ϕ. -1ϕ ∈ (q1'.term ╲ q1.term) ∧ ¬∃ϕ. -2ϕ ∈ (q2'.term ╲ q2.term)
            then AD := AD ∪ {<q0, q1', q2'>}
            else BD := BD ∪ {<q0, q1', q2'>}, CU := CU ∪ {<q0, q1', q2'>}
          fi
        CU := CU ∪ {q0} × (q1'.sub ╲ Resolved) ×(q2'.sub ╲ Resolved)
          ∪ {q0} × {q1'} ×(q2'.sub ╲ Resolved) ∪ {q0} × (q1'.sub ╲ Resolved) × {q2'}
        rof
    rof
    for <q0, q1, q2> ∈ CU
      CU := CU ╲ <q0, q1, q2>
      X := P-extend(q0, q1.prec1 ∪ q2.prec2)
      for <q0', _, _> ∈ X × {q1} × {q2} ╲ AU ╲ BU
        if ¬∃ϕ. 1ϕ or 2ϕ ∈ (q0'.term ╲ q0.term)
          then AU := AU ∪ <q0', q1, q2>
          else BU := BU ∪ <q0', q1, q2>, CD = CD ∪ <q0', q1, q2>
        fi
        CD := CD ∪ (q0'.sub ╲ Resolved) × {q1} × {q2}
      rof
    rof
  elihw
   δ := {}
  for <q0, q1, q2> ∈ AD ∪ AU
    R0 := {q | q.base = q0 ∧ q ∈ Resolved ∩ S}
    R1 := {q | q.base = q1 ∧ q ∈ Resolved ∩ P}
    R2 := {q | q.base = q2 ∧ q ∈ Resolved ∩ P}
    D := ({q0} ∪ R0) × ({q1} ∪ R1) × ({q2} ∪ R2)
    for <q0', q1', q2'> ∈ D
      δ := δ ∪ {(σ1∧…∧σn, q1', q2')→q0'} (σ1, … , σn ∈ q0'.term)
    rof
  rof
  F := { q | ∃Φ. Φ ⊆ q.term ∧ Φ ∈ expand(ϕ) }
  return <Q, F, δ>
corp
```

Figure 27

```
proc expand(Φ)
  Ψ := {}
  for φ ∈ Φ
    Φ := Φ \ {φ}
    case φ
      σ → Ψ := Ψ ∪ {σ}
      ψ ∧ ψ' → Φ := Φ ∪ {ψ, ψ'}
      ψ ∨ ψ' → return expand(Φ ∪ Ψ ∪ {ψ}) ∪ expand(Φ ∪ Ψ ∪ {ψ'})
      [m]ψ → Φ := Φ ∪ {ψ} ∪ {b[m]ψ | b ⊆ m}
      <m>ψ → return expand(Φ ∪ Ψ ∪ {ψ}) ∪ ∪_{b ⊆ m} expand(Φ ∪ Ψ ∪ {b<m>ψ, (b)})
      otherwise Ψ := Ψ ∪ {φ}
    esac
    fi
  rof
  return {Ψ}
corp
```

Figure 28

```
proc d-extend(q, Φ)
  if q = ε return {ε}
  X := {}
  Ξ := expand(Φ)
  if ∃ Ψ ∈ Ξ . Ψ ⊆ q.term then Ξ := {{}} fi
  for Ψ ∈ Ξ
    if ( q ∈ S ∧ d = S) ∨ ( q ∈ P ∧ d = P)
      then q' := <q.term ∪ Ψ , q.base>
      else q' := <q.term ∪ Ψ, q>
    fi
    X := X ∪ {q'}
    if ¬(q' ∈ Q) then
      Q := Q ∪ {q'}
      q'.succ1 := {ψ | 1 ψ ∈ q'.term ∖ q'.base.term }
      q'.succ2 := {ψ | 2 ψ ∈ q'.term ∖ q'.base.term }
      q'.prec1 := {ψ | -1 ψ ∈ q'.term ∖ q'.base.term}
      q'.prec2 := {ψ | -2 ψ ∈ q'.term ∖ q'.base.term}
      if d = S then
        S := S ∪ {q'}
        if q'.succ1 = q'.succ2 = {} then Resolved := Resolved ∪ {q'}
         else
          q'.succ1 := q'.succ1 ∪ q'.base.succ1
          q'.succ2 := q'.succ2 ∪ q'.base.succ2
          CD := CD ∪ {<q', q1, q2> | <q'.base, q1, q2> ∈ AU ∪ BU, q1, q2 ∈ Q}
         fi
      else
        P := P ∪ {q'}
        if q'.prec1 = q'.prec2 = {} then Resolved := Resolved ∪ {q'}
        else
          q'.prec1 := q'.prec1 ∪ q'.base.prec1
          q'.prec2 := q'.prec2 ∪ q'.base.prec2
          CU := CU ∪ {<q0, q', q''> | <q0, q'.base, q''.base>
                ∈ AD ∪ BD, q0 ∈ Q, q'' ∈ P ∖ Resolved}
                ∪ {<q0, q'', q'> | <q0, q''.base, q'.base>
                ∈ AD ∪ BD, q0 ∈ Q, q'' ∈ P ∖ Resolved}
                ∪ {<q0, q', q2> | <q0, q'.base, q2> ∈ AD ∪ BD, q0, q2 ∈ Q}
                ∪ {<q0, q1, q'> | <q0, q1, q'.base> ∈ AD ∪ BD, q0, q1 ∈ Q}
        fi
      fi
    fi
  rof
  return X
corp
```

Figure 29